Human Activity Recognition

July 8, 2019

1 HumanActivityRecognition

This project is to build a model that predicts the human activities such as Walking, Walking_Upstairs, Walking_Downstairs, Sitting, Standing or Laying.

This dataset is collected from 30 persons(referred as subjects in this dataset), performing different activities with a smartphone to their waists. The data is recorded with the help of sensors (accelerometer and Gyroscope) in that smartphone. This experiment was video recorded to label the data manually.

1.1 How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial linear acceleration'(*tAcc-XYZ*) from accelerometer and '3-axial angular velocity' (*tGyro-XYZ*) from Gyroscope with several variations.

> prefix 't' in those metrics denotes time.

> suffix 'XYZ' represents 3-axial signals in X , Y, and Z directions.

1.1.1 Feature names

1. These sensor signals are preprocessed by applying noise filters and then sampled in fixed-width windows(sliding windows) of 2.56 seconds each with 50% overlap. ie., each window has 128 readings.

2. From Each window, a feature vector was obtianed by calculating variables from the time and frequency domain. > In our dataset, each datapoint represents a window with different readings

3. The accelertion signal was saperated into Body and Gravity acceleration signals(***tBodyAcc-XYZ*** and ***tGravityAcc-XYZ***) using some low pass filter with corner frequecy of 0.3Hz.

4. After that, the body linear acceleration and angular velocity were derived in time to obtian *jerk signals* (***tBodyAccJerk-XYZ*** and ***tBodyGyroJerk-XYZ***).

5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This magnitudes are represented as features with names like *tBodyAccMag*, *tGravityAccMag*, *tBodyAccJerkMag*, *tBodyGyroMag* and *tBodyGyroJerkMag*.

6. Finally, We've got frequency domain signals from some of the available signals by applying a FFT (Fast Fourier Transform). These signals obtained were labeled with ***prefix 'f'*** just like original signals with ***prefix 't'***. These signals are labeled as ***fBodyAcc-XYZ***, ***fBodyGyroMag*** etc.,.

7. These are the signals that we got so far.

    - tBodyAcc-XYZ
    - tGravityAcc-XYZ
    - tBodyAccJerk-XYZ
    - tBodyGyro-XYZ
    - tBodyGyroJerk-XYZ
    - tBodyAccMag
    - tGravityAccMag
    - tBodyAccJerkMag
    - tBodyGyroMag
    - tBodyGyroJerkMag
    - fBodyAcc-XYZ
    - fBodyAccJerk-XYZ
    - fBodyGyro-XYZ
    - fBodyAccMag
    - fBodyAccJerkMag
    - fBodyGyroMag
    - fBodyGyroJerkMag

8. We can esitmate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recoreded so far.

    - ***mean()***: Mean value
    - ***std()***: Standard deviation
    - ***mad()***: Median absolute deviation
    - ***max()***: Largest value in array
    - ***min()***: Smallest value in array
    - ***sma()***: Signal magnitude area
    - ***energy()***: Energy measure. Sum of the squares divided by the number of values.
    - ***iqr()***: Interquartile range
    - ***entropy()***: Signal entropy
    - ***arCoeff()***: Autorregresion coefficients with Burg order equal to 4
    - ***correlation()***: correlation coefficient between two signals
    - ***maxInds()***: index of the frequency component with largest magnitude
    - ***meanFreq()***: Weighted average of the frequency components to obtain a mean frequency
    - ***skewness()***: skewness of the frequency domain signal
    - ***kurtosis()***: kurtosis of the frequency domain signal
    - ***bandsEnergy()***: Energy of a frequency interval within the 64 bins of the FFT of each window.
    - ***angle()***: Angle between to vectors.

9. We can obtain some other vectors by taking the average of signals in a single window sample. These are used on the angle() variable' '

    - gravityMean
    - tBodyAccMean
    - tBodyAccJerkMean
    - tBodyGyroMean
    - tBodyGyroJerkMean

1.1.2 Y_Labels(Encoded)

- In the dataset, Y_labels are represented as numbers from 1 to 6 as their identifiers.
  - WALKING as **1**
  - WALKING_UPSTAIRS as **2**
  - WALKING_DOWNSTAIRS as **3**
  - SITTING as **4**
  - STANDING as **5**
  - LAYING as **6**

1.2 Train and test data were saperated

- The readings from ***70%*** of the volunteers were taken as ***trianing data*** and remaining ***30%*** subjects recordings were taken for ***test data***

1.3 Data

- All the data is present in 'UCI_HAR_dataset/' folder in present working directory.
  - Feature names are present in 'UCI_HAR_dataset/features.txt'
  - ***Train Data***
    * 'UCI_HAR_dataset/train/X_train.txt'
    * 'UCI_HAR_dataset/train/subject_train.txt'
    * 'UCI_HAR_dataset/train/y_train.txt'
  - ***Test Data***
    * 'UCI_HAR_dataset/test/X_test.txt'
    * 'UCI_HAR_dataset/test/subject_test.txt'
    * 'UCI_HAR_dataset/test/y_test.txt'

1.4 Data Size :

27 MB

2 Quick overview of the dataset :

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) while performing the following 6 Activities.

1. Walking

2. WalkingUpstairs
3. WalkingDownstairs
4. Standing
5. Sitting
6. Lying.

- Readings are divided into a window of 2.56 seconds with 50% overlapping.
- Accelerometer readings are divided into gravity acceleration and body acceleration readings, which has x,y and z components each.
- Gyroscope readings are the measure of angular velocities which has x,y and z components.
- Jerk signals are calculated for BodyAcceleration readings.
- Fourier Transforms are made on the above time readings to obtain frequency readings.
- Now, on all the base signal readings., mean, max, mad, sma, arcoefficient, engerybands,entropy etc., are calculated for each window.
- We get a feature vector of 561 features and these features are given in the dataset.
- Each window of readings is a datapoint of 561 features.

2.1 Problem Framework

- 30 subjects(volunteers) data is randomly split to 70%(21) test and 30%(7) train data.
- Each datapoint corresponds one of the 6 Activities.

2.2 Problem Statement

- Given a new datapoint we have to predict the Activity

In [1]:
```
import numpy as np
import pandas as pd

# get the features from the file features.txt
features = list()
with open('UCI_HAR_Dataset/features.txt') as f:
    features = [line.split()[1] for line in f.readlines()]
print('No of Features: {}'.format(len(features)))
```

```
No of Features: 561
```

2.3 Obtain the train data

In [2]:
```
# get the data from txt files to pandas dataframe
X_train = pd.read_csv('UCI_HAR_Dataset/train/X_train.txt', delim_whitespace=True, heade

# add subject column to the dataframe
X_train['subject'] = pd.read_csv('UCI_HAR_Dataset/train/subject_train.txt', header=None

y_train = pd.read_csv('UCI_HAR_Dataset/train/y_train.txt', names=['Activity'], squeeze=
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

# put all columns in a single dataframe
train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.sample()
```

```
C:\Users\sirsh\Anaconda3\lib\site-packages\pandas\io\parsers.py:702: UserWarning: Duplicate nam
  return _read(filepath_or_buffer, kwds)
```

Out[2]:
```
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
5101           0.279996          -0.021338          -0.116085

      tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
5101          -0.99699         -0.968391         -0.988508         -0.997588

      tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
5101         -0.967537         -0.989509         -0.938963  ...

      angle(tBodyAccMean,gravity)  angle(tBodyAccJerkMean),gravityMean)  \
5101                     0.034435                              -0.043463

      angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMean)  \
5101                         -0.432307                              -0.655816

      angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)  \
5101              -0.64561              0.168566             -0.226414

      subject  Activity  ActivityName
5101       25         5      STANDING

[1 rows x 564 columns]
```

In [3]:
```
train.shape
```

Out[3]:
```
(7352, 564)
```

2.4 Obtain the test data

In [4]:
```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_Dataset/test/X_test.txt', delim_whitespace=True, header=N

# add subject column to the dataframe
X_test['subject'] = pd.read_csv('UCI_HAR_Dataset/test/subject_test.txt', header=None, s

# get y labels from the txt file
y_test = pd.read_csv('UCI_HAR_Dataset/test/y_test.txt', names=['Activity'], squeeze=Tru
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                        4:'SITTING', 5:'STANDING',6:'LAYING'})


# put all columns in a single dataframe
test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels
test.sample()
```

Out[4]:
```
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
1530           0.275578          -0.018537          -0.107052

      tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
1530         -0.996675         -0.978744         -0.977455          -0.99708

      tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
1530         -0.977893         -0.974458          -0.94359  ...

      angle(tBodyAccMean,gravity)  angle(tBodyAccJerkMean),gravityMean)  \
1530                     0.222092                             -0.051048

      angle(tBodyGyroMean,gravityMean)  angle(tBodyGyroJerkMean,gravityMean)  \
1530                          0.809108                             -0.556085

      angle(X,gravityMean)  angle(Y,gravityMean)  angle(Z,gravityMean)  \
1530             -0.749686              0.238888              0.121501

      subject  Activity  ActivityName
1530       13         5      STANDING

[1 rows x 564 columns]
```

In [5]:
```
test.shape
```

Out[5]:
```
(2947, 564)
```

3 Data Cleaning

3.1 1. Check for Duplicates

```
In [6]: print('No of duplicates in train: {}'.format(sum(train.duplicated())))
        print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

```
No of duplicates in train: 0
No of duplicates in test : 0
```

3.2 2. Checking for NaN/null values

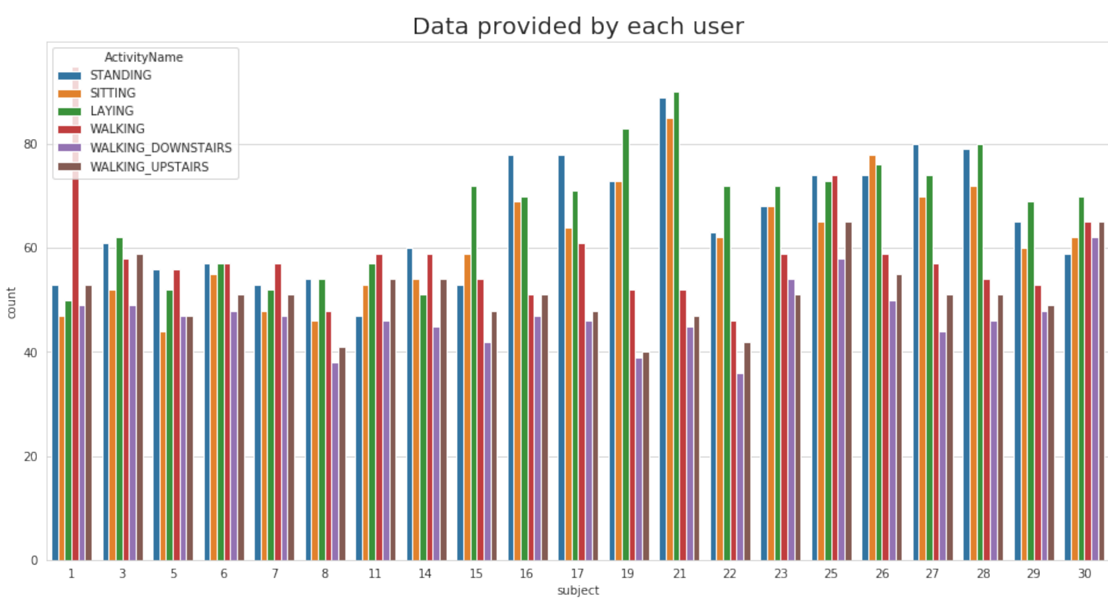
```
In [7]: print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
        print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

```
We have 0 NaN/Null values in train
We have 0 NaN/Null values in test
```

3.3 3. Check for data imbalance

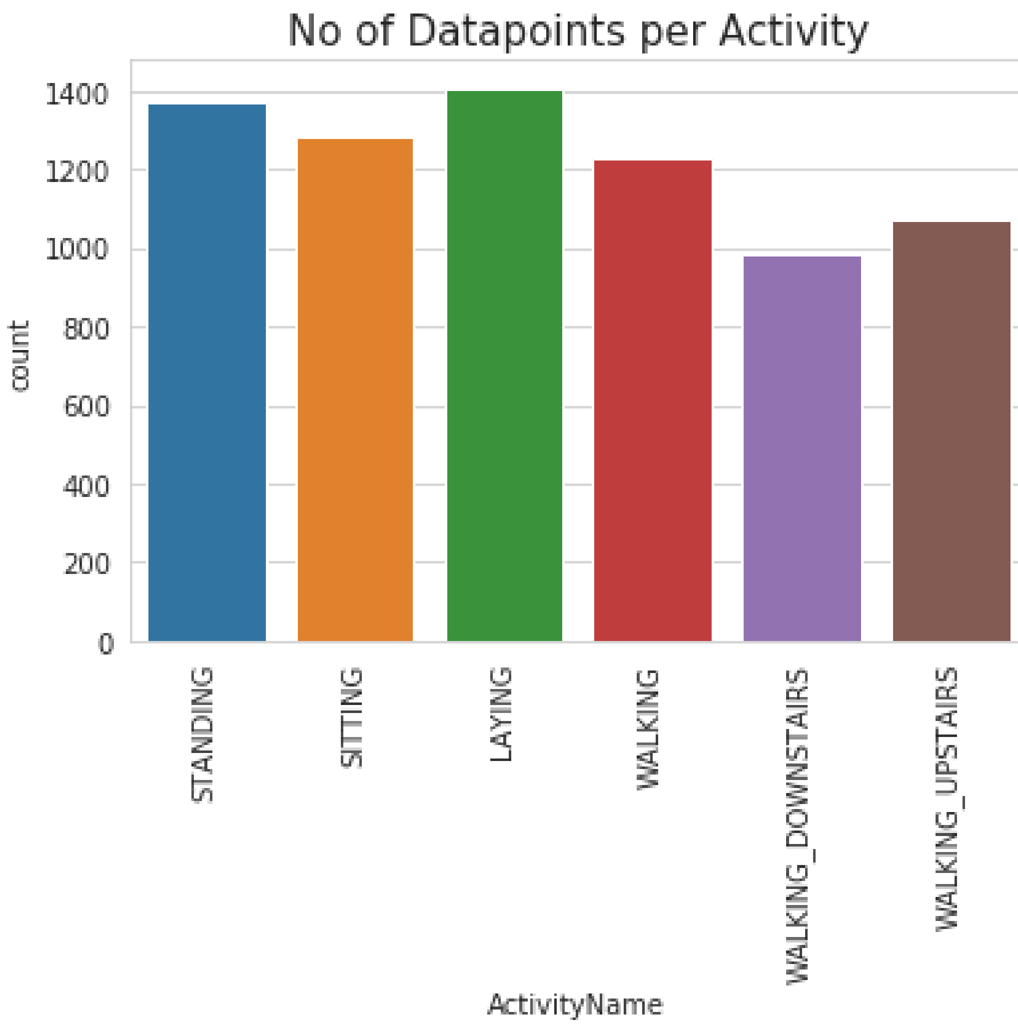
```
In [8]: import matplotlib.pyplot as plt
        import seaborn as sns

        sns.set_style('whitegrid')
        plt.rcParams['font.family'] = 'Dejavu Sans'
```

```
In [9]: plt.figure(figsize=(16,8))
        plt.title('Data provided by each user', fontsize=20)
        sns.countplot(x='subject',hue='ActivityName', data = train)
        plt.show()
```



We have got almost same number of reading from all the subjects

```
In [10]: plt.title('No of Datapoints per Activity', fontsize=15)
         sns.countplot(train.ActivityName)
         plt.xticks(rotation=90)
         plt.show()
```



3.3.1 Observation

Our data is well balanced (almost)

3.4 4. Changing feature names

In [11]:
```
columns = train.columns

# Removing '()' from column names
columns = columns.str.replace('[()]','')
columns = columns.str.replace('[-]', '')
columns = columns.str.replace('[,]','')

train.columns = columns
test.columns = columns

test.columns
```

Out[11]:
```
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
       'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
       'tBodyAccmadZ', 'tBodyAccmaxX',
       ...
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
       'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
       'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
       'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)
```

3.5 5. Save this dataframe in a csv files

In [12]:
```
train.to_csv('UCI_HAR_Dataset/csv_files/train.csv', index=False)
test.to_csv('UCI_HAR_Dataset/csv_files/test.csv', index=False)
```

4 Exploratory Data Analysis

"Without domain knowledge EDA has no meaning, without EDA a problem has no soul."

4.0.1 1. Featuring Engineering from Domain Knowledge

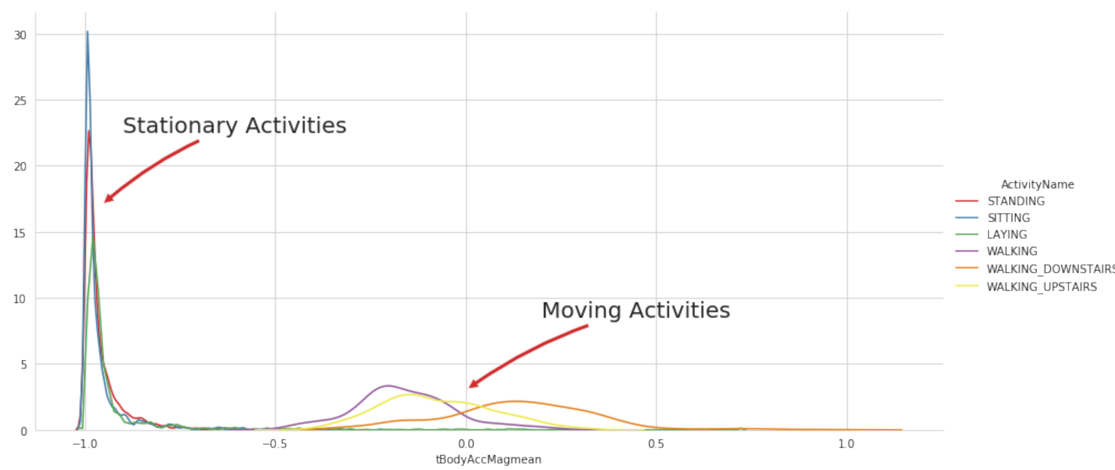
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

4.0.2 2. Stationary and Moving activities are completely different

In [13]:
```
sns.set_palette("Set1", desat=0.80)
facetgrid = sns.FacetGrid(train, hue='ActivityName', size=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", xy=(-0.956,17), xytext=(-0.9, 23), size=20,\
            va='center', ha='left',\
```

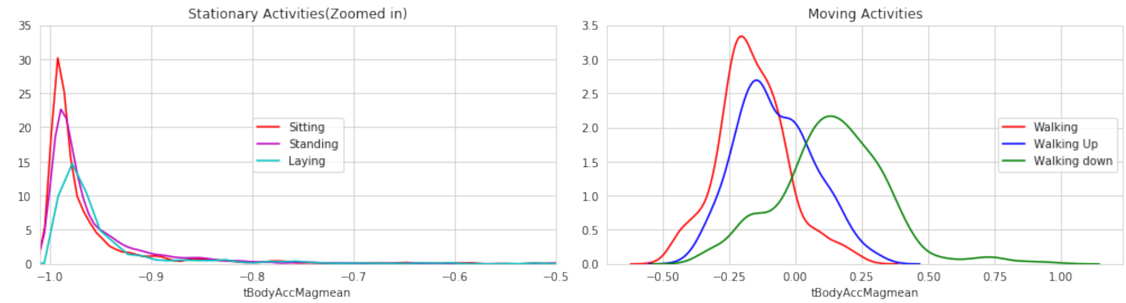
```
                arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

plt.annotate("Moving Activities", xy=(0,3), xytext=(0.2, 9), size=20,\
                va='center', ha='left',\
                arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))
plt.show()
```

```
/home/ae/anaconda3/lib/python3.7/site-packages/seaborn/axisgrid.py:230: UserWarning: The `size
  warnings.warn(msg, UserWarning)
```



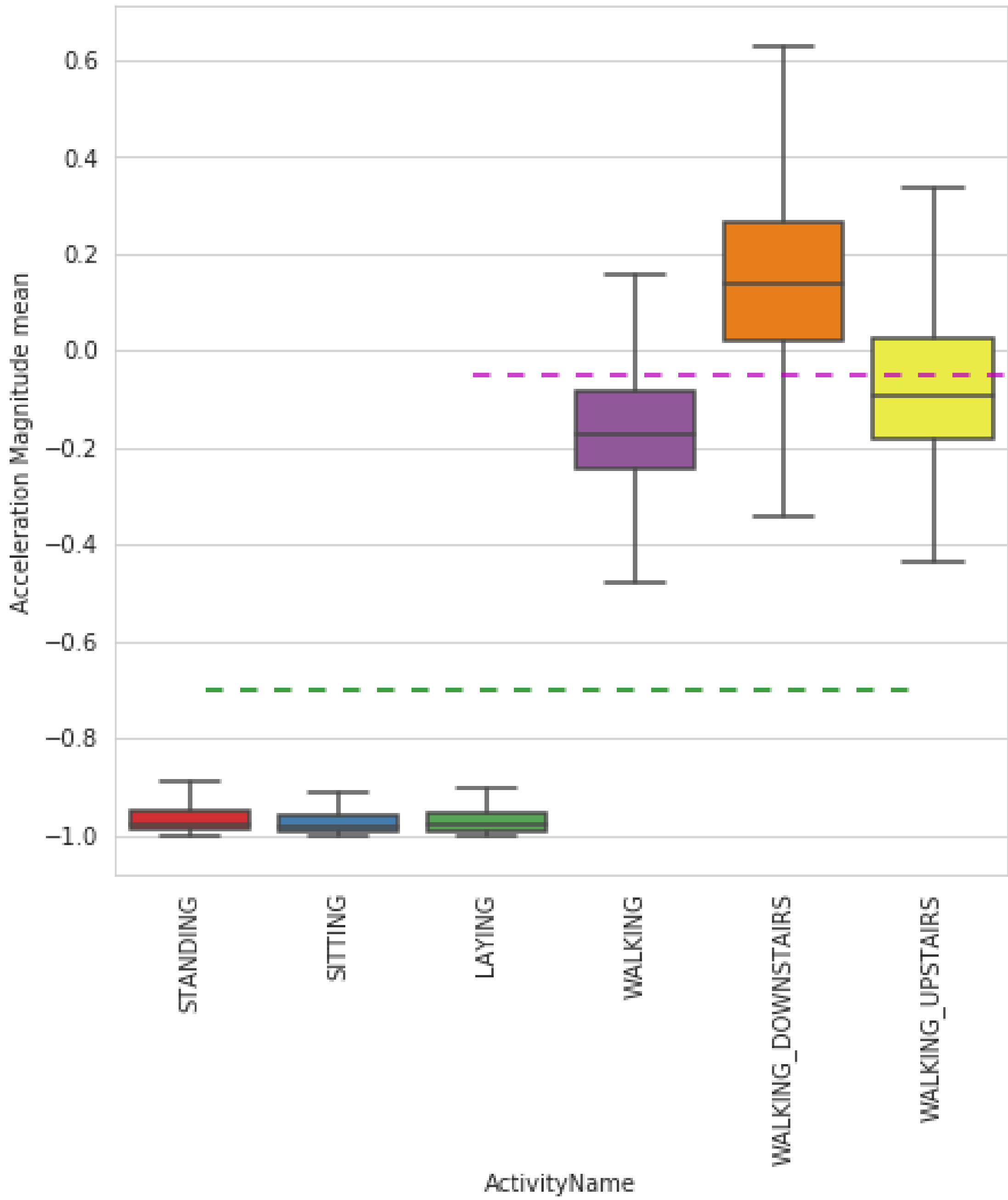
In [14]:
```
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

plt.figure(figsize=(14,7))
plt.subplot(2,2,1)
plt.title('Stationary Activities(Zoomed in)')
sns.distplot(df4['tBodyAccMagmean'],color = 'r',hist = False, label = 'Sitting')
sns.distplot(df5['tBodyAccMagmean'],color = 'm',hist = False,label = 'Standing')
sns.distplot(df6['tBodyAccMagmean'],color = 'c',hist = False, label = 'Laying')
plt.axis([-1.01, -0.5, 0, 35])
plt.legend(loc='center')

plt.subplot(2,2,2)
plt.title('Moving Activities')
sns.distplot(df1['tBodyAccMagmean'],color = 'red',hist = False, label = 'Walking')
```

```
sns.distplot(df2['tBodyAccMagmean'],color = 'blue',hist = False,label = 'Walking Up')
sns.distplot(df3['tBodyAccMagmean'],color = 'green',hist = False, label = 'Walking do
plt.legend(loc='center right')


plt.tight_layout()
plt.show()
```



4.0.3 3. Magnitude of an acceleration can saperate it well

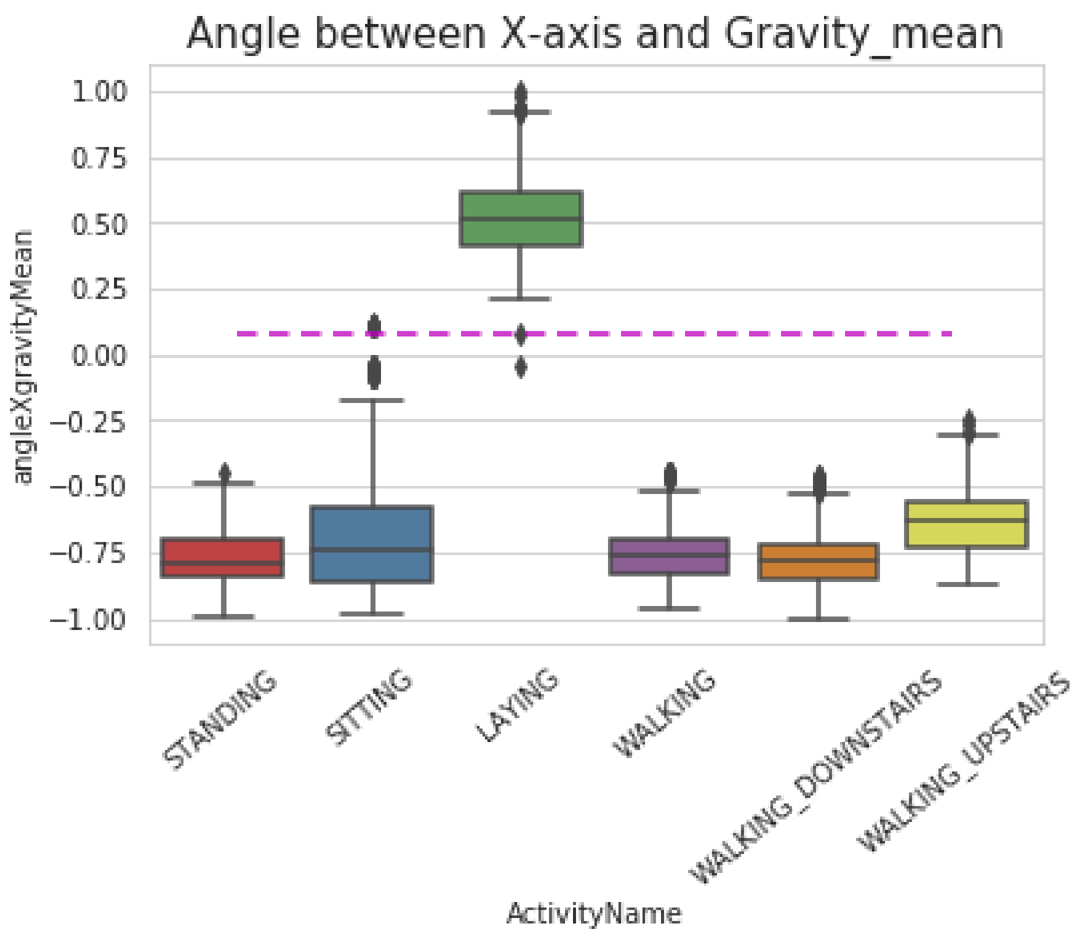
```
In [15]: plt.figure(figsize=(7,7))
         sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, satura
         plt.ylabel('Acceleration Magnitude mean')
         plt.axhline(y=-0.7, xmin=0.1, xmax=0.9,dashes=(5,5), c='g')
         plt.axhline(y=-0.05, xmin=0.4, dashes=(5,5), c='m')
         plt.xticks(rotation=90)
         plt.show()
```



__ Observations__: - If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying. - If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or WalkingUpstairs. - If tAccMean > 0.0 then the Activity is WalkingDownstairs. - We can classify 75% the Acitivity labels with some errors.

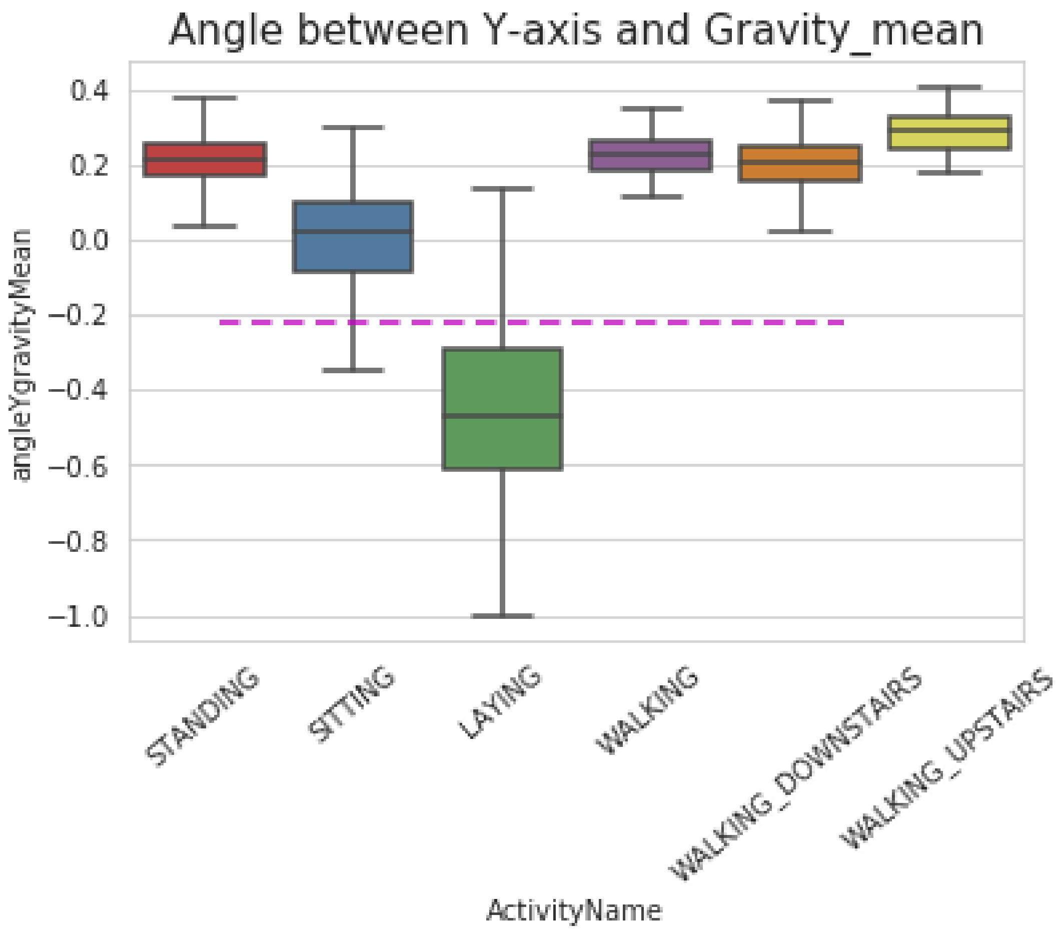
4.0.4 4. Position of GravityAccelerationComponants also matters

```
In [16]: sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
         plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
         plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
         plt.xticks(rotation = 40)
         plt.show()
```



__ Observations__: * If angleX,gravityMean > 0 then Activity is Laying. * We can classify all datapoints belonging to Laying activity with just a single if else statement.

```
In [17]: sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, showfliers=False)
         plt.title('Angle between Y-axis and Gravity_mean', fontsize=15)
         plt.xticks(rotation = 40)
         plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
         plt.show()
```



5 Apply t-sne on the data

In [18]:
```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
```

In [19]:
```
# performs t-sne with different perplexity values and their repective plots..

def perform_tsne(X_data, y_data, perplexities, n_iter=1000, img_name_prefix='t-sne'):

    for index,perplexity in enumerate(perplexities):
        # perform t-sne
        print('\nperforming tsne with perplexity {} and with {} iterations at max'.fo
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)
        print('Done..')
```

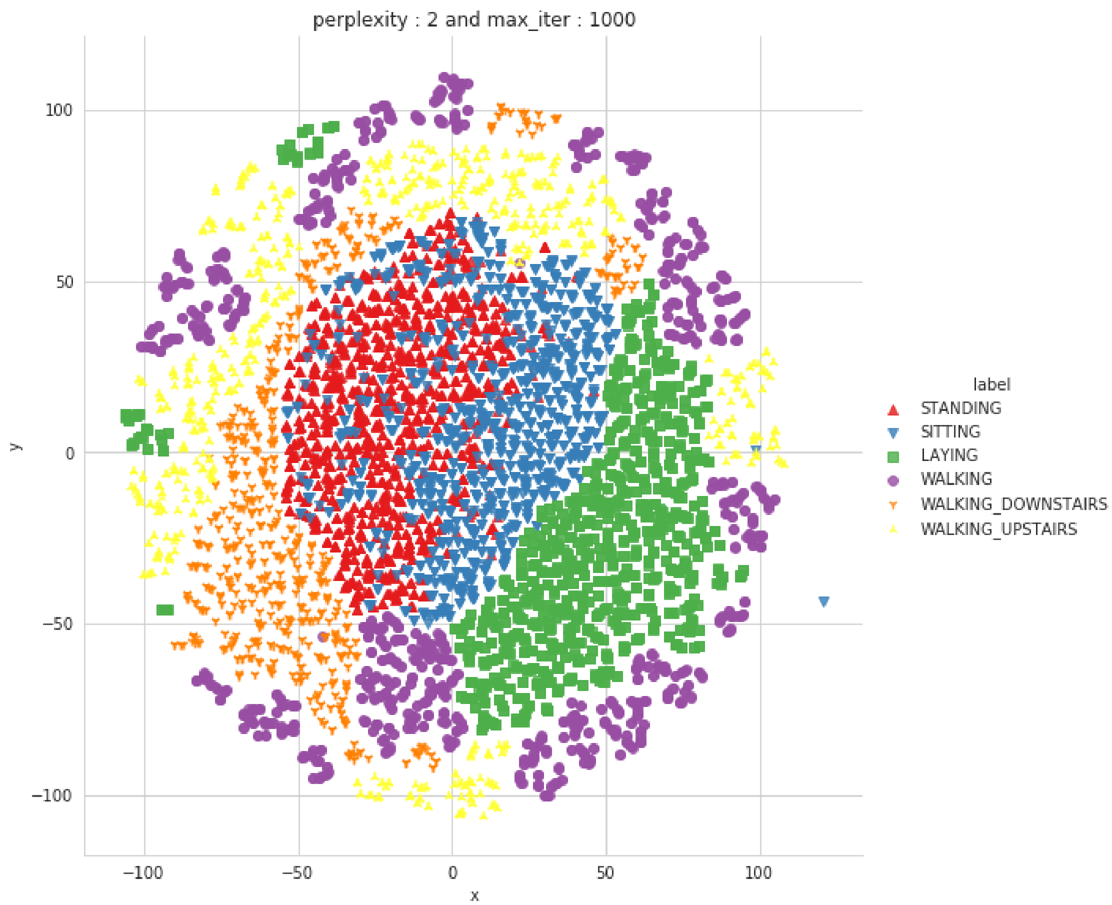
```
            # prepare the data for seaborn
            print('Creating plot for this t-sne visualization..')
            df = pd.DataFrame({'x':X_reduced[:,0], 'y':X_reduced[:,1] ,'label':y_data})

            # draw the plot in appropriate place in the grid
            sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,\
                       palette="Set1",markers=['^','v','s','o', '1','2'])
            plt.title("perplexity : {} and max_iter : {}".format(perplexity, n_iter))
            img_name = img_name_prefix + '_perp_{}_iter_{}.png'.format(perplexity, n_iter)
            print('saving this plot as image in present working directory...')
            plt.savefig(img_name)
            plt.show()
            print('Done')
```

In [29]:
```
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```

```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.117s...
[t-SNE] Computed neighbors for 7352 samples in 26.631s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.635855
[t-SNE] Computed conditional probabilities in 0.034s
[t-SNE] Iteration 50: error = 124.7129745, gradient norm = 0.0251482 (50 iterations in 4.331s)
[t-SNE] Iteration 100: error = 106.8463669, gradient norm = 0.0287980 (50 iterations in 2.652s)
[t-SNE] Iteration 150: error = 100.6308212, gradient norm = 0.0186865 (50 iterations in 1.913s)
[t-SNE] Iteration 200: error = 97.2790833, gradient norm = 0.0144918 (50 iterations in 1.708s)
[t-SNE] Iteration 250: error = 94.9964447, gradient norm = 0.0111065 (50 iterations in 1.700s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 94.996445
[t-SNE] Iteration 300: error = 4.1111989, gradient norm = 0.0015574 (50 iterations in 1.544s)
[t-SNE] Iteration 350: error = 3.2055898, gradient norm = 0.0009988 (50 iterations in 1.422s)
[t-SNE] Iteration 400: error = 2.7764344, gradient norm = 0.0007158 (50 iterations in 1.602s)
[t-SNE] Iteration 450: error = 2.5130441, gradient norm = 0.0005693 (50 iterations in 1.396s)
[t-SNE] Iteration 500: error = 2.3302758, gradient norm = 0.0004719 (50 iterations in 1.435s)
[t-SNE] Iteration 550: error = 2.1924589, gradient norm = 0.0004129 (50 iterations in 1.410s)
[t-SNE] Iteration 600: error = 2.0833056, gradient norm = 0.0003690 (50 iterations in 1.533s)
[t-SNE] Iteration 650: error = 1.9937783, gradient norm = 0.0003299 (50 iterations in 1.455s)
[t-SNE] Iteration 700: error = 1.9181801, gradient norm = 0.0003023 (50 iterations in 1.448s)
```

```
[t-SNE] Iteration 750: error = 1.8531532, gradient norm = 0.0002758 (50 iterations in 1.429s)
[t-SNE] Iteration 800: error = 1.7966599, gradient norm = 0.0002563 (50 iterations in 1.428s)
[t-SNE] Iteration 850: error = 1.7468235, gradient norm = 0.0002400 (50 iterations in 1.441s)
[t-SNE] Iteration 900: error = 1.7023505, gradient norm = 0.0002245 (50 iterations in 1.450s)
[t-SNE] Iteration 950: error = 1.6621462, gradient norm = 0.0002114 (50 iterations in 1.434s)
[t-SNE] Iteration 1000: error = 1.6257638, gradient norm = 0.0002022 (50 iterations in 1.442s)
[t-SNE] KL divergence after 1000 iterations: 1.625764
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

```
/home/ae/anaconda3/lib/python3.7/site-packages/seaborn/regression.py:546: UserWarning: The `si
  warnings.warn(msg, UserWarning)
```

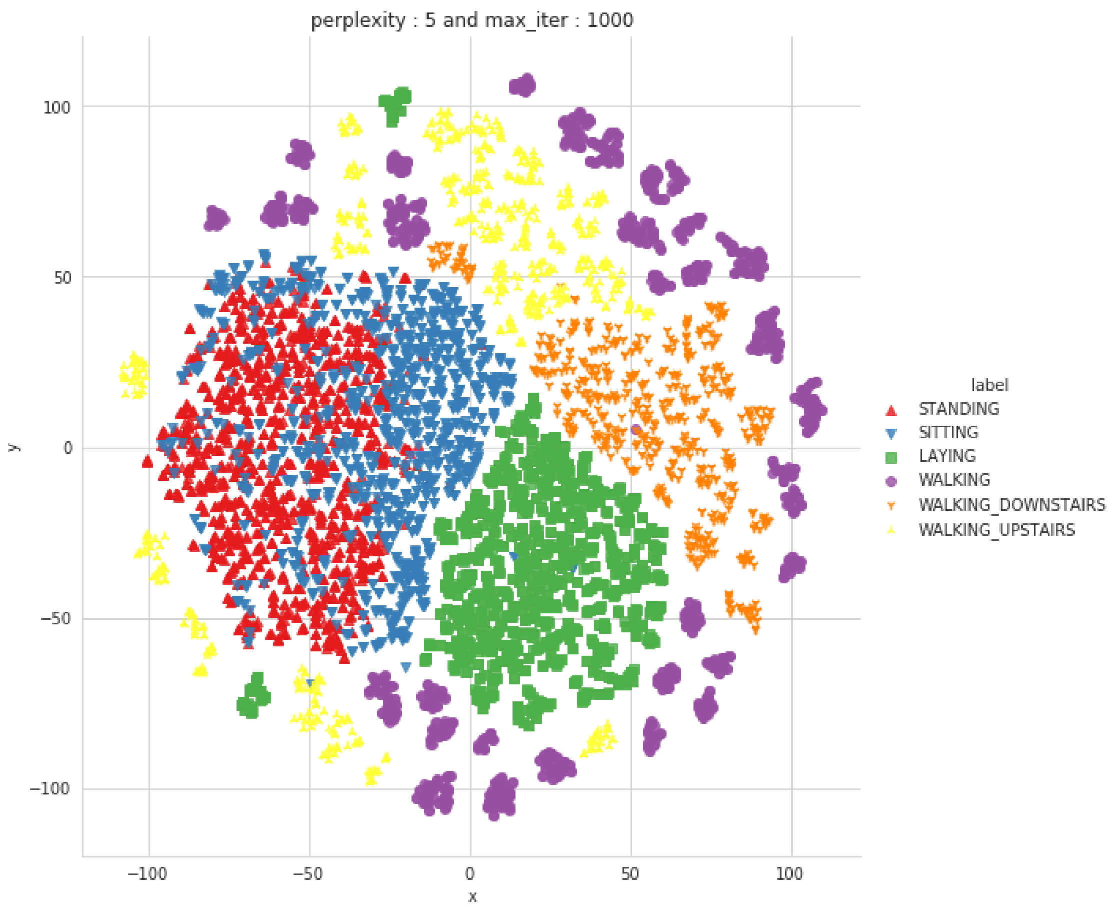



```
Done
```

```
performing tsne with perplexity 5 and with 1000 iterations at max
```

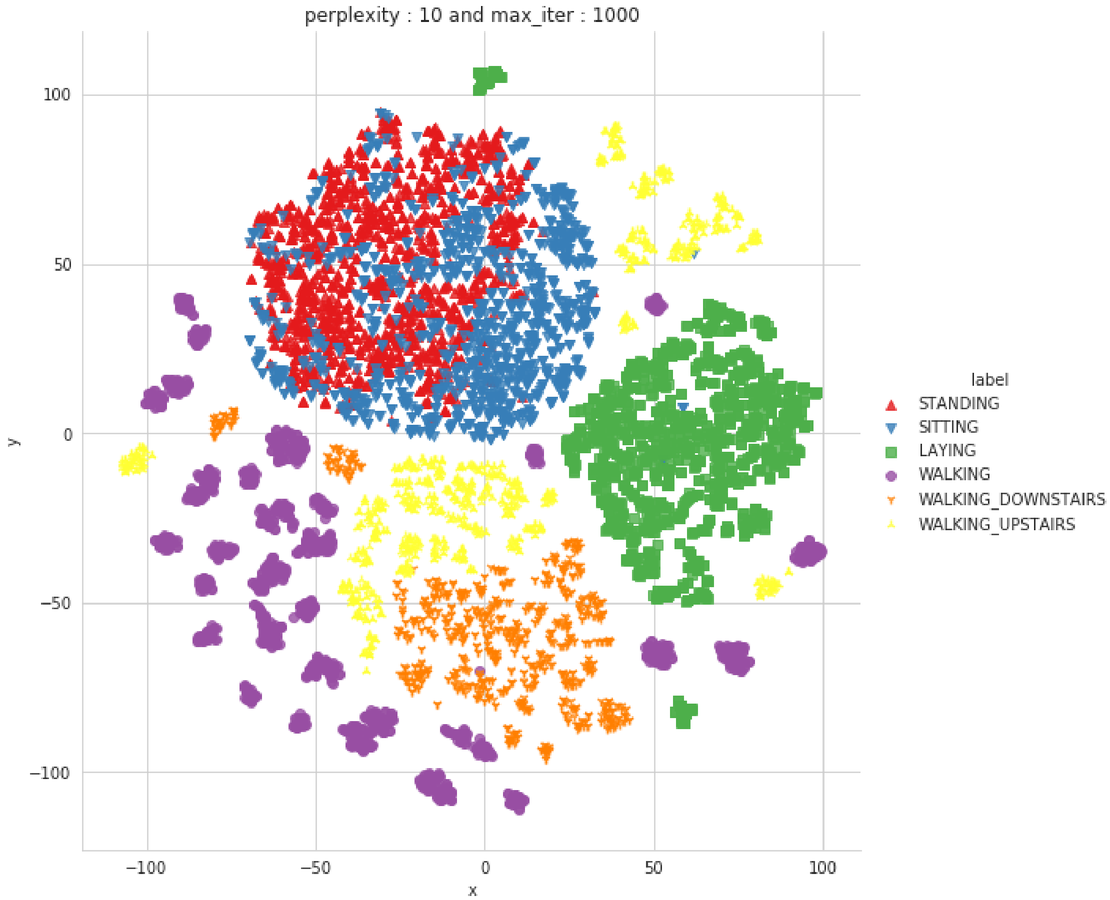
```
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.112s...
[t-SNE] Computed neighbors for 7352 samples in 28.261s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.961265
[t-SNE] Computed conditional probabilities in 0.061s
[t-SNE] Iteration 50: error = 113.9727936, gradient norm = 0.0266489 (50 iterations in 2.817s)
[t-SNE] Iteration 100: error = 97.7394791, gradient norm = 0.0157285 (50 iterations in 1.759s)
[t-SNE] Iteration 150: error = 93.4704056, gradient norm = 0.0094903 (50 iterations in 1.514s)
[t-SNE] Iteration 200: error = 91.4397659, gradient norm = 0.0073299 (50 iterations in 1.528s)
[t-SNE] Iteration 250: error = 90.1913681, gradient norm = 0.0054473 (50 iterations in 1.467s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.191368
[t-SNE] Iteration 300: error = 3.5743632, gradient norm = 0.0014544 (50 iterations in 1.480s)
[t-SNE] Iteration 350: error = 2.8167980, gradient norm = 0.0007482 (50 iterations in 1.379s)
[t-SNE] Iteration 400: error = 2.4360650, gradient norm = 0.0005249 (50 iterations in 1.371s)
[t-SNE] Iteration 450: error = 2.2184384, gradient norm = 0.0004056 (50 iterations in 1.399s)
[t-SNE] Iteration 500: error = 2.0734482, gradient norm = 0.0003315 (50 iterations in 1.418s)
[t-SNE] Iteration 550: error = 1.9677753, gradient norm = 0.0002835 (50 iterations in 1.401s)
[t-SNE] Iteration 600: error = 1.8866595, gradient norm = 0.0002480 (50 iterations in 1.407s)
[t-SNE] Iteration 650: error = 1.8214889, gradient norm = 0.0002201 (50 iterations in 1.520s)
[t-SNE] Iteration 700: error = 1.7677324, gradient norm = 0.0001988 (50 iterations in 1.426s)
[t-SNE] Iteration 750: error = 1.7221799, gradient norm = 0.0001826 (50 iterations in 1.412s)
[t-SNE] Iteration 800: error = 1.6832911, gradient norm = 0.0001664 (50 iterations in 1.400s)
[t-SNE] Iteration 850: error = 1.6492078, gradient norm = 0.0001536 (50 iterations in 1.498s)
[t-SNE] Iteration 900: error = 1.6193261, gradient norm = 0.0001425 (50 iterations in 1.456s)
[t-SNE] Iteration 950: error = 1.5928975, gradient norm = 0.0001341 (50 iterations in 1.424s)
[t-SNE] Iteration 1000: error = 1.5693035, gradient norm = 0.0001249 (50 iterations in 1.475s)
[t-SNE] KL divergence after 1000 iterations: 1.569304
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

```
/home/ae/anaconda3/lib/python3.7/site-packages/seaborn/regression.py:546: UserWarning: The `siz
  warnings.warn(msg, UserWarning)
```



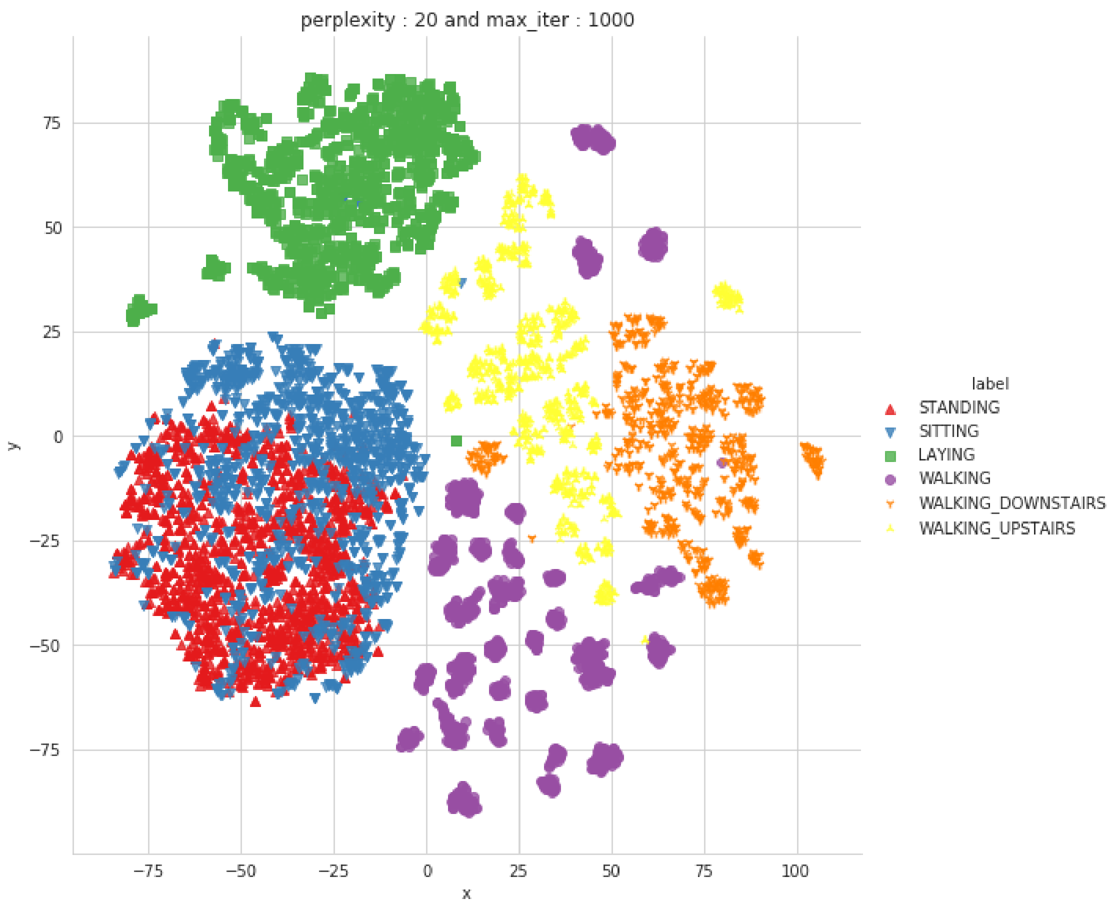

```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.115s...
[t-SNE] Computed neighbors for 7352 samples in 29.213s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.133828
[t-SNE] Computed conditional probabilities in 0.116s
[t-SNE] Iteration 50: error = 105.9907532, gradient norm = 0.0168286 (50 iterations in 3.701s)
[t-SNE] Iteration 100: error = 91.0411987, gradient norm = 0.0108318 (50 iterations in 1.825s)
[t-SNE] Iteration 150: error = 87.4613495, gradient norm = 0.0050138 (50 iterations in 1.584s)
```

```
[t-SNE] Iteration 200: error = 86.1291809, gradient norm = 0.0042984 (50 iterations in 1.553s)
[t-SNE] Iteration 250: error = 85.3850098, gradient norm = 0.0029340 (50 iterations in 1.496s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.385010
[t-SNE] Iteration 300: error = 3.1372325, gradient norm = 0.0014026 (50 iterations in 1.584s)
[t-SNE] Iteration 350: error = 2.4914017, gradient norm = 0.0006515 (50 iterations in 1.429s)
[t-SNE] Iteration 400: error = 2.1710777, gradient norm = 0.0004278 (50 iterations in 1.536s)
[t-SNE] Iteration 450: error = 1.9864763, gradient norm = 0.0003170 (50 iterations in 1.398s)
[t-SNE] Iteration 500: error = 1.8681291, gradient norm = 0.0002522 (50 iterations in 1.402s)
[t-SNE] Iteration 550: error = 1.7848518, gradient norm = 0.0002121 (50 iterations in 1.465s)
[t-SNE] Iteration 600: error = 1.7222220, gradient norm = 0.0001808 (50 iterations in 1.517s)
[t-SNE] Iteration 650: error = 1.6732755, gradient norm = 0.0001596 (50 iterations in 1.467s)
[t-SNE] Iteration 700: error = 1.6338762, gradient norm = 0.0001422 (50 iterations in 1.492s)
[t-SNE] Iteration 750: error = 1.6012387, gradient norm = 0.0001301 (50 iterations in 1.534s)
[t-SNE] Iteration 800: error = 1.5738049, gradient norm = 0.0001178 (50 iterations in 1.414s)
[t-SNE] Iteration 850: error = 1.5502882, gradient norm = 0.0001106 (50 iterations in 1.529s)
[t-SNE] Iteration 900: error = 1.5301794, gradient norm = 0.0001014 (50 iterations in 1.560s)
[t-SNE] Iteration 950: error = 1.5125302, gradient norm = 0.0000956 (50 iterations in 1.486s)
[t-SNE] Iteration 1000: error = 1.4969953, gradient norm = 0.0000920 (50 iterations in 1.562s)
[t-SNE] KL divergence after 1000 iterations: 1.496995
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...


/home/ae/anaconda3/lib/python3.7/site-packages/seaborn/regression.py:546: UserWarning: The `si
  warnings.warn(msg, UserWarning)
```



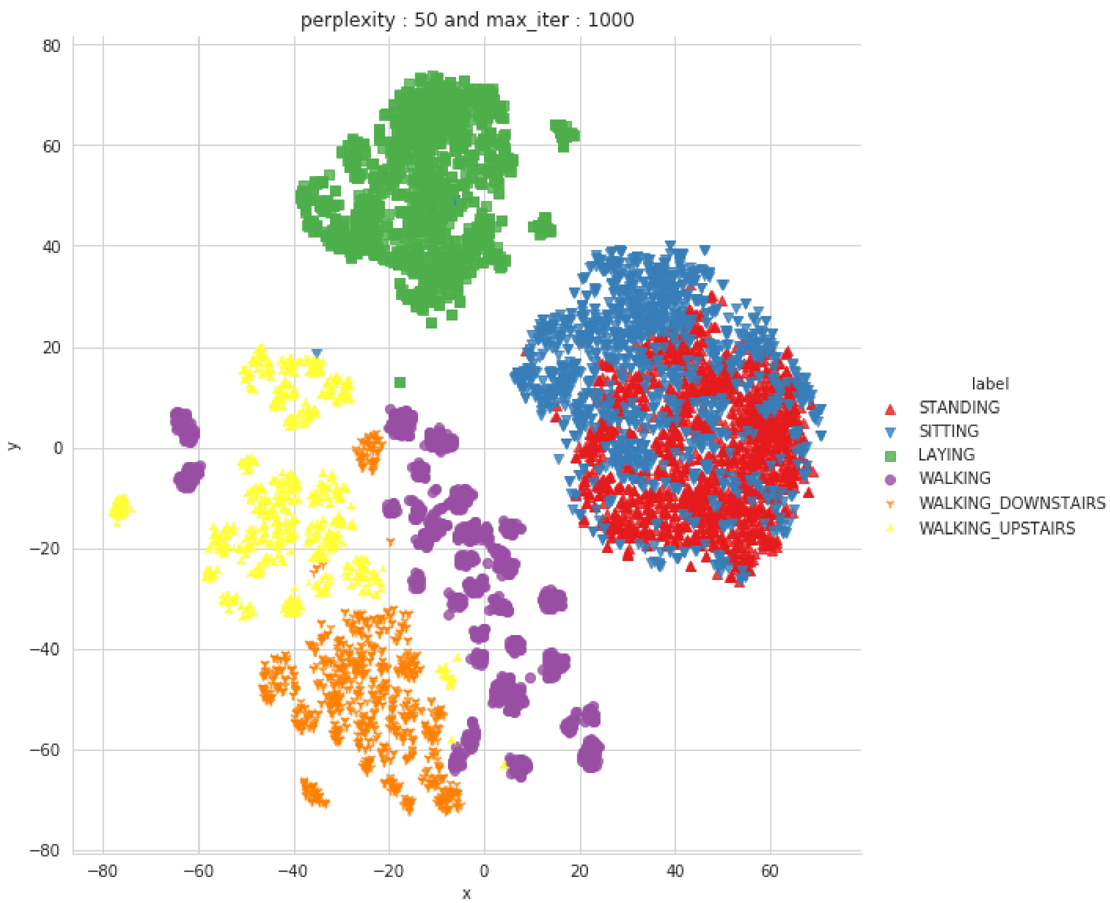

```
Done

performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.121s...
[t-SNE] Computed neighbors for 7352 samples in 28.129s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.274335
[t-SNE] Computed conditional probabilities in 0.216s
[t-SNE] Iteration 50: error = 95.7582703, gradient norm = 0.0337219 (50 iterations in 4.132s)
[t-SNE] Iteration 100: error = 83.9358444, gradient norm = 0.0070196 (50 iterations in 2.397s)
[t-SNE] Iteration 150: error = 81.8789139, gradient norm = 0.0040086 (50 iterations in 1.979s)
```

```
[t-SNE] Iteration 200: error = 81.1673355, gradient norm = 0.0026776 (50 iterations in 1.943s)
[t-SNE] Iteration 250: error = 80.7847672, gradient norm = 0.0016252 (50 iterations in 2.064s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.784767
[t-SNE] Iteration 300: error = 2.7092786, gradient norm = 0.0013063 (50 iterations in 1.910s)
[t-SNE] Iteration 350: error = 2.1744046, gradient norm = 0.0005757 (50 iterations in 1.645s)
[t-SNE] Iteration 400: error = 1.9245238, gradient norm = 0.0003485 (50 iterations in 1.633s)
[t-SNE] Iteration 450: error = 1.7776188, gradient norm = 0.0002502 (50 iterations in 1.624s)
[t-SNE] Iteration 500: error = 1.6836761, gradient norm = 0.0001920 (50 iterations in 1.622s)
[t-SNE] Iteration 550: error = 1.6193535, gradient norm = 0.0001590 (50 iterations in 1.660s)
[t-SNE] Iteration 600: error = 1.5728641, gradient norm = 0.0001337 (50 iterations in 1.622s)
[t-SNE] Iteration 650: error = 1.5378749, gradient norm = 0.0001181 (50 iterations in 1.624s)
[t-SNE] Iteration 700: error = 1.5104412, gradient norm = 0.0001059 (50 iterations in 1.639s)
[t-SNE] Iteration 750: error = 1.4884633, gradient norm = 0.0000961 (50 iterations in 1.628s)
[t-SNE] Iteration 800: error = 1.4709184, gradient norm = 0.0000916 (50 iterations in 1.655s)
[t-SNE] Iteration 850: error = 1.4569169, gradient norm = 0.0000856 (50 iterations in 1.655s)
[t-SNE] Iteration 900: error = 1.4452990, gradient norm = 0.0000801 (50 iterations in 1.678s)
[t-SNE] Iteration 950: error = 1.4354850, gradient norm = 0.0000767 (50 iterations in 1.576s)
[t-SNE] Iteration 1000: error = 1.4272671, gradient norm = 0.0000737 (50 iterations in 1.594s)
[t-SNE] KL divergence after 1000 iterations: 1.427267
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...


/home/ae/anaconda3/lib/python3.7/site-packages/seaborn/regression.py:546: UserWarning: The `si
  warnings.warn(msg, UserWarning)
```




```
Done

performing tsne with perplexity 50 and with 1000 iterations at max
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.109s...
[t-SNE] Computed neighbors for 7352 samples in 29.738s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.437672
[t-SNE] Computed conditional probabilities in 0.432s
[t-SNE] Iteration 50: error = 86.6766357, gradient norm = 0.0183803 (50 iterations in 3.192s)
[t-SNE] Iteration 100: error = 75.5323868, gradient norm = 0.0044215 (50 iterations in 2.592s)
[t-SNE] Iteration 150: error = 74.5760803, gradient norm = 0.0021047 (50 iterations in 2.166s)
```

```
[t-SNE] Iteration 200: error = 74.2252121, gradient norm = 0.0018476 (50 iterations in 2.186s)
[t-SNE] Iteration 250: error = 74.0481491, gradient norm = 0.0011039 (50 iterations in 2.127s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.048149
[t-SNE] Iteration 300: error = 2.1546106, gradient norm = 0.0011807 (50 iterations in 2.136s)
[t-SNE] Iteration 350: error = 1.7561660, gradient norm = 0.0004920 (50 iterations in 1.966s)
[t-SNE] Iteration 400: error = 1.5873977, gradient norm = 0.0002810 (50 iterations in 1.939s)
[t-SNE] Iteration 450: error = 1.4933531, gradient norm = 0.0001916 (50 iterations in 1.963s)
[t-SNE] Iteration 500: error = 1.4333763, gradient norm = 0.0001412 (50 iterations in 1.948s)
[t-SNE] Iteration 550: error = 1.3920431, gradient norm = 0.0001123 (50 iterations in 2.025s)
[t-SNE] Iteration 600: error = 1.3628286, gradient norm = 0.0000948 (50 iterations in 2.120s)
[t-SNE] Iteration 650: error = 1.3414272, gradient norm = 0.0000826 (50 iterations in 1.851s)
[t-SNE] Iteration 700: error = 1.3259615, gradient norm = 0.0000759 (50 iterations in 1.915s)
[t-SNE] Iteration 750: error = 1.3146623, gradient norm = 0.0000689 (50 iterations in 1.910s)
[t-SNE] Iteration 800: error = 1.3058467, gradient norm = 0.0000634 (50 iterations in 2.039s)
[t-SNE] Iteration 850: error = 1.2985491, gradient norm = 0.0000614 (50 iterations in 1.878s)
[t-SNE] Iteration 900: error = 1.2926712, gradient norm = 0.0000586 (50 iterations in 1.926s)
[t-SNE] Iteration 950: error = 1.2876254, gradient norm = 0.0000564 (50 iterations in 1.861s)
[t-SNE] Iteration 1000: error = 1.2834342, gradient norm = 0.0000545 (50 iterations in 1.896s)
[t-SNE] KL divergence after 1000 iterations: 1.283434
Done..
Creating plot for this t-sne visualization..


/home/ae/anaconda3/lib/python3.7/site-packages/seaborn/regression.py:546: UserWarning: The `si
  warnings.warn(msg, UserWarning)


saving this plot as image in present working directory...
```




```
Done
```

```
In [20]: import numpy as np
         import pandas as pd
```

5.1 Obtain the train and test data

```
In [21]: train = pd.read_csv('UCI_HAR_Dataset/csv_files/train.csv')
         test = pd.read_csv('UCI_HAR_Dataset/csv_files/test.csv')
         print(train.shape, test.shape)
         train.head(3)
```

```
(7352, 564) (2947, 564)
```

Out[21]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	\
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	

```
2       0.279653      -0.019467      -0.113462     -0.995380     -0.967187

   tBodyAccstdZ  tBodyAccmadX  tBodyAccmadY  tBodyAccmadZ  tBodyAccmaxX  ...  \
0     -0.913526     -0.995112     -0.983185     -0.923527     -0.934724  ...
1     -0.960322     -0.998807     -0.974914     -0.957686     -0.943068  ...
2     -0.978944     -0.996520     -0.963668     -0.977469     -0.938692  ...

   angletBodyAccMeangravity  angletBodyAccJerkMeangravityMean  \
0                 -0.112754                          0.030400
1                  0.053477                         -0.007435
2                 -0.118559                          0.177899

   angletBodyGyroMeangravityMean  angletBodyGyroJerkMeangravityMean  \
0                      -0.464761                          -0.018446
1                      -0.732626                           0.703511
2                       0.100699                           0.808529

   angleXgravityMean  angleYgravityMean  angleZgravityMean  subject  Activity  \
0          -0.841247           0.179941          -0.058627        1         5
1          -0.844788           0.180289          -0.054317        1         5
2          -0.848933           0.180637          -0.049118        1         5

   ActivityName
0      STANDING
1      STANDING
2      STANDING

[3 rows x 564 columns]
```

In [22]:
```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.ActivityName
```

In [23]:
```
# get X_test and y_test from test csv file
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.ActivityName
```

In [24]:
```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
```

```
X_train and y_train : ((7352, 561),(7352,))
X_test  and y_test  : ((2947, 561),(2947,))
```

6 Let's model with our data

6.0.1 Labels that are useful in plotting confusion matrix

In [25]:
```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAI
```

6.0.2 Function to plot the confusion matrix

```
In [26]: import itertools
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix
         plt.rcParams["font.family"] = 'DejaVu Sans'

         def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=90)
             plt.yticks(tick_marks, classes)

             fmt = '.2f' if normalize else 'd'
             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, format(cm[i, j], fmt),
                          horizontalalignment="center",
                          color="white" if cm[i, j] > thresh else "black")

             plt.tight_layout()
             plt.ylabel('True label')
             plt.xlabel('Predicted label')
```

6.0.3 Generic function to run any model specified

```
In [27]: from datetime import datetime
         def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=
                          print_cm=True, cm_cmap=plt.cm.Greens):


             # to store results at various phases
             results = dict()

             # time at which model starts training
             train_start_time = datetime.now()
             print('training the model..')
             model.fit(X_train, y_train)
             print('Done \n \n')
```

```
train_end_time = datetime.now()
results['training_time'] =  train_end_time - train_start_time
print('training_time(HH:MM:SS.ms) - {}\n\n'.format(results['training_time']))


# predict test data
print('Predicting test data')
test_start_time = datetime.now()
y_pred = model.predict(X_test)
test_end_time = datetime.now()
print('Done \n \n')
results['testing_time'] = test_end_time - test_start_time
print('testing time(HH:MM:SS:ms) - {}\n\n'.format(results['testing_time']))
results['predicted'] = y_pred


# calculate overall accuracty of the model
accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
# store accuracy in results
results['accuracy'] = accuracy
print('---------------------')
print('|      Accuracy      |')
print('---------------------')
print('\n    {}\n\n'.format(accuracy))


# confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
results['confusion_matrix'] = cm
if print_cm:
    print('--------------------')
    print('| Confusion Matrix |')
    print('--------------------')
    print('\n {}'.format(cm))

# plot confusin matrix
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(cm, classes=class_labels, normalize=True, title='Normalized
plt.show()

# get classification report
print('-------------------------')
print('| Classifiction Report |')
print('-------------------------')
classification_report = metrics.classification_report(y_test, y_pred)
# store report in results
results['classification_report'] = classification_report
```

```
        print(classification_report)

        # add the trained  model to the results
        results['model'] = model

        return results
```

6.0.4 Method to print the gridsearch Attributes

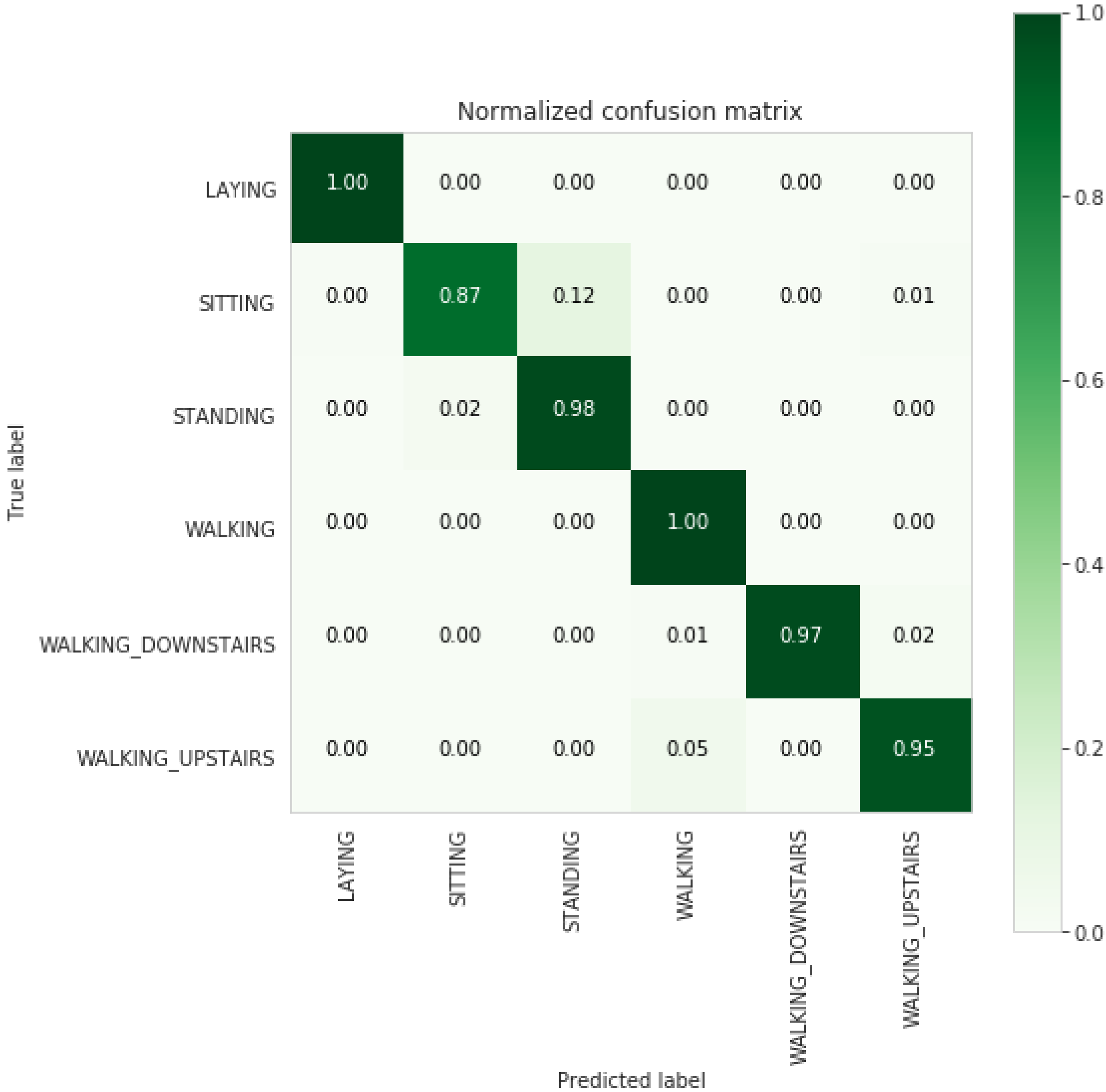
```
In [28]: def print_grid_search_attributes(model):
             # Estimator that gave highest score among all the estimators formed in GridSearch
             print('--------------------------')
             print('|      Best Estimator     |')
             print('--------------------------')
             print('\n\t{}\n'.format(model.best_estimator_))


             # parameters that gave best results while performing grid search
             print('--------------------------')
             print('|     Best parameters     |')
             print('--------------------------')
             print('\tParameters of best estimator : \n\n\t{}\n'.format(model.best_params_))


             #  number of cross validation splits
             print('---------------------------------')
             print('|   No of CrossValidation sets   |')
             print('--------------------------------')
             print('\n\tTotal numbre of cross validation sets: {}\n'.format(model.n_splits_))


             # Average cross validated score of the best estimator, from the Grid Search
             print('--------------------------')
             print('|        Best Score       |')
             print('--------------------------')
             print('\n\tAverage Cross Validate scores of best estimator : \n\n\t{}\n'.format(mo
```

7 1. Logistic Regression with Grid Search

```
In [29]: from sklearn import linear_model
         from sklearn import metrics

         from sklearn.model_selection import GridSearchCV

In [30]: # start Grid search
         parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
```

```
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test, y_test,
```

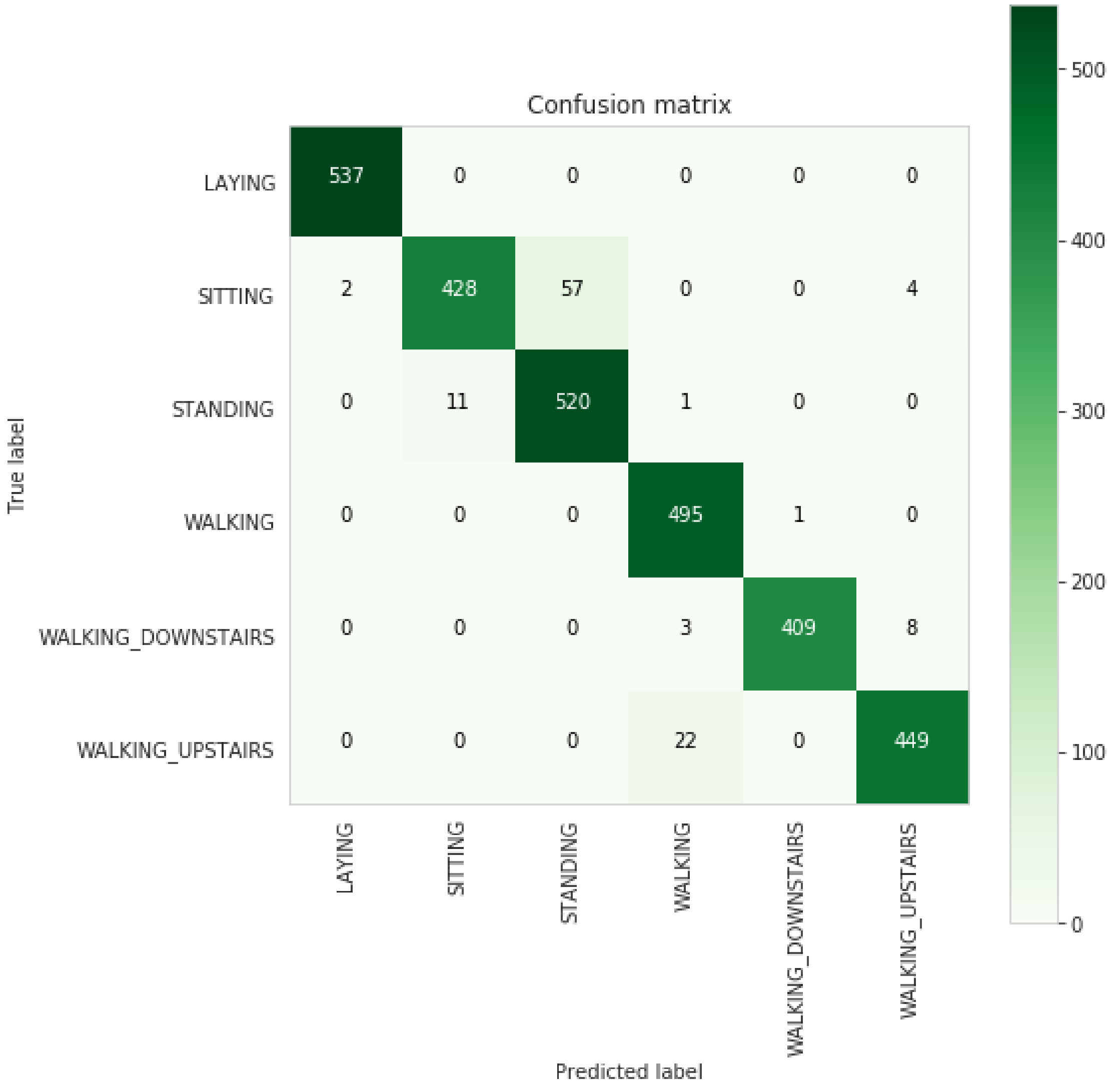
```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  1.1min finished
/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWar
  FutureWarning)
/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:460: FutureWar
  "this warning.", FutureWarning)


Done


training_time(HH:MM:SS.ms) - 0:01:23.263947


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.010340


---------------------
|      Accuracy      |
---------------------

    0.9630132337970818


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 428  57   0   0   4]
 [  0  11 520   1   0   0]
 [  0   0   0 495   1   0]
 [  0   0   0   3 409   8]
 [  0   0   0  22   0 449]]
```




```
--------------------------
| Classifiction Report |
--------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.95      1.00      0.97       496
WALKING_DOWNSTAIRS       1.00      0.97      0.99       420
  WALKING_UPSTAIRS       0.97      0.95      0.96       471

         micro avg       0.96      0.96      0.96      2947
```

```
   macro avg       0.97      0.96      0.96      2947
weighted avg       0.96      0.96      0.96      2947
```

```
In [44]: plt.figure(figsize=(8,8))
         plt.grid(b=False)
         plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes=labels, cmap=p
         plt.show()
```




```
In [45]: # observe the attributes of the model
         print_grid_search_attributes(log_reg_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
```

```
--------------------------

        LogisticRegression(C=30, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

--------------------------
|      Best parameters     |
--------------------------
        Parameters of best estimator :

        {'C': 30, 'penalty': 'l2'}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9461371055495104
```

8 2. Linear SVC with GridSearch

```
In [46]: from sklearn.svm import LinearSVC
```

```
In [47]: parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
         lr_svc = LinearSVC(tol=0.00005)
         lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
         lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test, cla
```

```
training the model..
Fitting 3 folds for each of 6 candidates, totalling 18 fits


/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2053: FutureWa
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 out of  18 | elapsed:   17.2s remaining:    4.9s
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   17.6s finished
```

```
/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:931: ConvergenceWarning: Lil
  "the number of iterations.", ConvergenceWarning)


Done


training_time(HH:MM:SS.ms) - 0:00:20.811854


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.002533


---------------------
|      Accuracy      |
---------------------

    0.9664065151001018


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 427  58   0   0   4]
 [  0  10 521   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   2 413   5]
 [  0   0   0  17   0 454]]
```


```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.98      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.96      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.96      0.97       471

         micro avg       0.97      0.97      0.97      2947
```

```
   macro avg       0.97      0.97      0.97      2947
weighted avg       0.97      0.97      0.97      2947
```

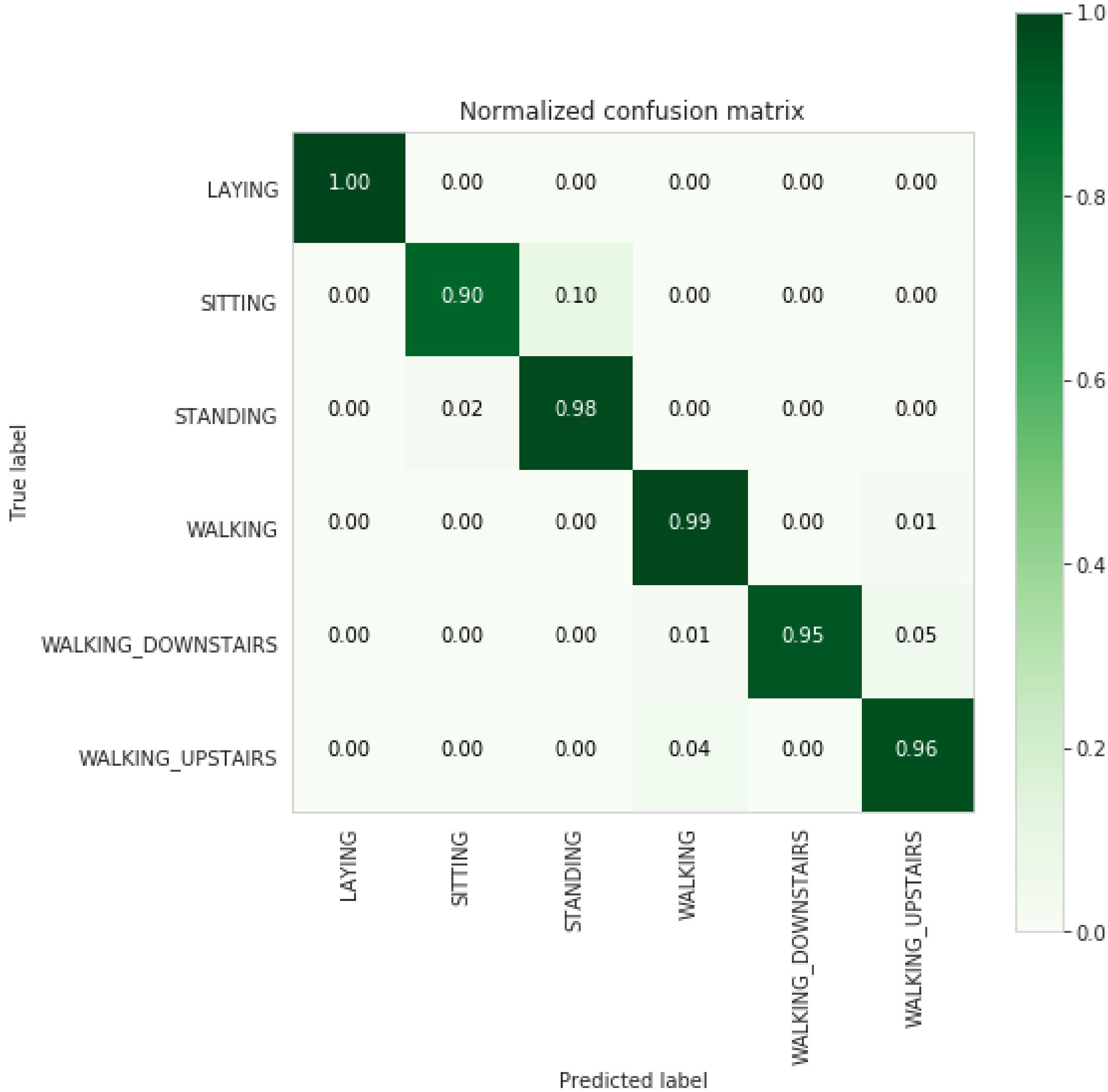
```
In [48]: print_grid_search_attributes(lr_svc_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

        LinearSVC(C=1, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=5e-05,
     verbose=0)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'C': 1}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9462731229597389
```

9 3. Kernel SVM with GridSearch

```
In [49]: from sklearn.svm import SVC
         parameters = {'C':[2,8,16],\
                       'gamma': [ 0.0078125, 0.125, 2]}
         rbf_svm = SVC(kernel='rbf')
         rbf_svm_grid = GridSearchCV(rbf_svm,param_grid=parameters, n_jobs=-1)
         rbf_svm_grid_results = perform_model(rbf_svm_grid, X_train, y_train, X_test, y_test, c
```

```
training the model..


/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2053: FutureWa
  warnings.warn(CV_WARNING, FutureWarning)
/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_
  "timeout or by a memory leak.", UserWarning


Done


training_time(HH:MM:SS.ms) - 0:04:10.604861


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:02.015974


---------------------
|      Accuracy      |
---------------------

    0.9626739056667798


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 441  48   0   0   2]
 [  0  12 520   0   0   0]
 [  0   0   0 489   2   5]
 [  0   0   0   4 397  19]
 [  0   0   0  17   1 453]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.90      0.93       491
          STANDING       0.92      0.98      0.95       532
           WALKING       0.96      0.99      0.97       496
WALKING_DOWNSTAIRS       0.99      0.95      0.97       420
  WALKING_UPSTAIRS       0.95      0.96      0.95       471

         micro avg       0.96      0.96      0.96      2947
```

```
    macro avg       0.96      0.96      0.96      2947
 weighted avg       0.96      0.96      0.96      2947
```

```
In [50]: print_grid_search_attributes(rbf_svm_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

        SVC(C=16, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.0078125, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'C': 16, 'gamma': 0.0078125}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9440968443960827
```

10 4. Decision Trees with GridSearchCV

```
In [53]: from sklearn.tree import DecisionTreeClassifier
         parameters = {'max_depth':np.arange(3,10,2)}
         dt = DecisionTreeClassifier()
         dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
         dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_label
         print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..

/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2053: FutureWa
  warnings.warn(CV_WARNING, FutureWarning)

Done

training_time(HH:MM:SS.ms) - 0:00:05.092272

Predicting test data
Done

testing time(HH:MM:SS:ms) - 0:00:00.002446

---------------------
|      Accuracy      |
---------------------

    0.8632507634882932

--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 385 106   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 470  18   8]
 [  0   0   0  15 344  61]
 [  0   0   0  73  29 369]]
```


```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.81      0.78      0.79       491
          STANDING       0.81      0.83      0.82       532
           WALKING       0.84      0.95      0.89       496
WALKING_DOWNSTAIRS       0.88      0.82      0.85       420
  WALKING_UPSTAIRS       0.84      0.78      0.81       471

         micro avg       0.86      0.86      0.86      2947
```

```
   macro avg       0.86      0.86      0.86      2947
weighted avg       0.86      0.86      0.86      2947

--------------------------
|      Best Estimator     |
--------------------------

        DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'max_depth': 7}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.8378672470076169
```

11 5. Random Forest Classifier with GridSearch

```
In [54]: from sklearn.ensemble import RandomForestClassifier
         params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}
         rfc = RandomForestClassifier()
         rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
         rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_test, class_la
         print_grid_search_attributes(rfc_grid_results['model'])
```

```
training the model..
```

```
/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2053: FutureWa
  warnings.warn(CV_WARNING, FutureWarning)


Done


training_time(HH:MM:SS.ms) - 0:02:23.761178


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.015416


---------------------
|      Accuracy      |
---------------------

    0.9060061079063454


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 424  67   0   0   0]
 [  0  59 473   0   0   0]
 [  0   0   0 480   9   7]
 [  0   0   0  35 336  49]
 [  0   0   0  45   6 420]]
```


```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.88      0.86      0.87       491
          STANDING       0.88      0.89      0.88       532
           WALKING       0.86      0.97      0.91       496
WALKING_DOWNSTAIRS       0.96      0.80      0.87       420
  WALKING_UPSTAIRS       0.88      0.89      0.89       471

         micro avg       0.91      0.91      0.91      2947
```

```
   macro avg       0.91      0.90      0.90      2947
weighted avg       0.91      0.91      0.91      2947

--------------------------
|      Best Estimator     |
--------------------------

        RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=7, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'max_depth': 7, 'n_estimators': 50}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.9147170837867247
```

12 6. Gradient Boosted Decision Trees With GridSearch

```
In [55]: from sklearn.ensemble import GradientBoostingClassifier
         param_grid = {'max_depth': np.arange(5,8,1), \
                      'n_estimators':np.arange(130,170,10)}
         gbdt = GradientBoostingClassifier()
         gbdt_grid = GridSearchCV(gbdt, param_grid=param_grid, n_jobs=-1)
         gbdt_grid_results = perform_model(gbdt_grid, X_train, y_train, X_test, y_test, class_l
         print_grid_search_attributes(gbdt_grid_results['model'])
```

```
training the model..

/home/ae/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2053: FutureWa
  warnings.warn(CV_WARNING, FutureWarning)

Done

training_time(HH:MM:SS.ms) - 0:23:54.298271

Predicting test data
Done

testing time(HH:MM:SS:ms) - 0:00:00.048356

---------------------
|      Accuracy      |
---------------------

    0.9212758737699356

--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 394  96   0   0   1]
 [  0  38 494   0   0   0]
 [  0   0   0 483   7   6]
 [  0   0   0  10 374  36]
 [  0   1   0  31   6 433]]
```


```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.80      0.85       491
          STANDING       0.84      0.93      0.88       532
           WALKING       0.92      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.89      0.93       420
  WALKING_UPSTAIRS       0.91      0.92      0.91       471

         micro avg       0.92      0.92      0.92      2947
```

```
    macro avg       0.92      0.92      0.92      2947
 weighted avg       0.92      0.92      0.92      2947

--------------------------
|      Best Estimator     |
--------------------------

        GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=5,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=130,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'max_depth': 5, 'n_estimators': 130}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.905195865070729
```

13 7. Comparing all models

In [56]:
```
print('\n                     Accuracy     Error')
print('                     ----------   --------')
print('Logistic Regression : {:.04}%      {:.04}%'.format(log_reg_grid_results['accu
                                                 100-(log_reg_grid_results['accuracy
```

```
print('Linear SVC          : {:.04}%       {:.04}% '.format(lr_svc_grid_results['accu
                                                     100-(lr_svc_grid_results['accu

print('rbf SVM classifier  : {:.04}%      {:.04}% '.format(rbf_svm_grid_results['accu
                                                       100-(rbf_svm_grid_results['a

print('DecisionTree        : {:.04}%      {:.04}% '.format(dt_grid_results['accuracy']
                                                     100-(dt_grid_results['accuracy

print('Random Forest       : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy
                                                        100-(rfc_grid_results['accu
print('GradientBoosting DT : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy
                                                     100-(rfc_grid_results['accura
```

```
                     Accuracy     Error
                     ----------   --------
Logistic Regression : 96.3%       3.699%
Linear SVC          : 96.64%       3.359%
rbf SVM classifier  : 96.27%      3.733%
DecisionTree        : 86.33%      13.67%
Random Forest       : 90.6%      9.399%
GradientBoosting DT : 90.6%      9.399%
```

We can choose ***Logistic regression*** or ***Linear SVC*** or ***rbf SVM***.

14 Conclusion :

In the real world, domain-knowledge, EDA and feature-engineering matter most.

15 Importing Libraries for Deep Learning

In [3]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}
```

```
# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

In [5]:
```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [6]:
```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [7]:
```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))
```

In [8]:
```
def load_y(subset):
    """
```

```
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

In [9]:
```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

In [10]:
```
# Importing tensorflow
np.random.seed(2)
import tensorflow as tf
tf.set_random_seed(2)
```

In [11]:
```
# Configuring a session
session_conf = tf.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [12]:
```
# Import Keras
from keras import backend as K
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

```
Using TensorFlow backend.
```

In [13]:
```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM, BatchNormalization
from keras.layers.core import Dense, Dropout
import keras
```

In [14]:
```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [15]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

```
C:\Users\sirsh\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: FutureWarning: Method .as_
  if sys.path[0] == '':
```

In [29]:
```
timesteps = len(X_train[0])
inp_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

- Defining the Architecture of LSTM

In [30]:
```
# Initializing parameters
epochs = 30
batch_size = 128
```

15.1 Creating a MLP

In [69]:
```
from keras.layers import Reshape, Input, Flatten
from keras import Input, Model, Sequential
from keras.layers import Conv1D, MaxPooling1D, Concatenate, Activation, Dropout, Flat

input_shape = Input(shape=(timesteps, input_dim))

#Model 1
model_1 = Conv1D(64,(4,), padding='same', activation='relu')(input_shape)
model_1 = MaxPooling1D((2,), strides=(1,), padding='same')(model_1)
model_1 = Dropout(0.5)(model_1)

model_1 = Conv1D(128,(4,), padding='same', activation='relu')(model_1)
model_1 = MaxPooling1D((2,), strides=(1,), padding='same')(model_1)

model_1 = Conv1D(256,(4,), padding='same', activation='relu')(model_1)
model_1 = MaxPooling1D((2,), strides=(1,), padding='same')(model_1)

model_1 = Conv1D(32,(4,), padding='same', activation='relu')(model_1)
model_1 = MaxPooling1D((2,), strides=(1,), padding='same')(model_1)

model_1 = Flatten()(model_1)
```

```
    model_1 = Dropout(0.8)(model_1)

    #Model 2
    model_2 = LSTM(64, kernel_initializer=keras.initializers.glorot_normal(seed=None), re
    model_2 = Dropout(0.8)(model_2)
    model_2 = LSTM(128, kernel_initializer=keras.initializers.glorot_normal(seed=None))(m
    model_2 = Dropout(0.6)(model_2)

    merged = keras.layers.concatenate([model_1, model_2], axis=1)

    out = Dense(64, activation='relu')(merged)
    out = Dropout(0.7)(merged)
    out = Dense(n_classes, activation='softmax')(out)

    model = Model(input_shape, out)
    model.summary()

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']
```

```
W0708 21:48:04.952948  4740 deprecation_wrapper.py:119] From C:\Users\sirsh\Anaconda3\lib\site



__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_9 (InputLayer)            (None, 128, 9)       0
__________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 128, 64)      2368        input_9[0][0]
__________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 128, 64)      0           conv1d_1[0][0]
__________________________________________________________________________________________
dropout_64 (Dropout)            (None, 128, 64)      0           max_pooling1d_1[0][0]
__________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 128, 128)     32896       dropout_64[0][0]
__________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 128, 128)     0           conv1d_2[0][0]
__________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 128, 256)     131328      max_pooling1d_2[0][0]
__________________________________________________________________________________________
max_pooling1d_3 (MaxPooling1D)  (None, 128, 256)     0           conv1d_3[0][0]
__________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, 128, 32)      32800       max_pooling1d_3[0][0]
__________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 128, 64)      18944       input_9[0][0]
__________________________________________________________________________________________
max_pooling1d_4 (MaxPooling1D)  (None, 128, 32)      0           conv1d_4[0][0]
__________________________________________________________________________________________
```

```
dropout_66 (Dropout)          (None, 128, 64)      0            lstm_1[0][0]
____________________________________________________________________________________________________
flatten_5 (Flatten)           (None, 4096)         0            max_pooling1d_4[0][0]
____________________________________________________________________________________________________
lstm_2 (LSTM)                 (None, 128)          98816        dropout_66[0][0]
____________________________________________________________________________________________________
dropout_65 (Dropout)          (None, 4096)         0            flatten_5[0][0]
____________________________________________________________________________________________________
dropout_67 (Dropout)          (None, 128)          0            lstm_2[0][0]
____________________________________________________________________________________________________
concatenate_1 (Concatenate)   (None, 4224)         0            dropout_65[0][0]
                                                                dropout_67[0][0]
____________________________________________________________________________________________________
dropout_68 (Dropout)          (None, 4224)         0            concatenate_1[0][0]
____________________________________________________________________________________________________
dense_82 (Dense)              (None, 6)            25350        dropout_68[0][0]
====================================================================================================
Total params: 342,502
Trainable params: 342,502
Non-trainable params: 0
____________________________________________________________________________________________________
```

In [70]:
```
from keras.callbacks import ModelCheckpoint

# Training the model
checkpoint = ModelCheckpoint('weights_lstm_cnn.hdf5',\
                             verbose=1, monitor='val_acc',save_best_only=True, mode='a


history1 = model.fit(X_train,
          Y_train,
          batch_size=128,
          validation_data=(X_test, Y_test),
          epochs=50,
            callbacks=[checkpoint])
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 79s 11ms/step - loss: 1.2015 - acc: 0.4810 - val_l

Epoch 00001: val_acc improved from -inf to 0.58738, saving model to weights_lstm_cnn.hdf5
Epoch 2/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.6707 - acc: 0.6813 - val_l

Epoch 00002: val_acc improved from 0.58738 to 0.59993, saving model to weights_lstm_cnn.hdf5
Epoch 3/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.5217 - acc: 0.7688 - val_l
```

```
Epoch 00003: val_acc improved from 0.59993 to 0.74211, saving model to weights_lstm_cnn.hdf5
Epoch 4/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.4501 - acc: 0.8138 - val_l

Epoch 00004: val_acc improved from 0.74211 to 0.79844, saving model to weights_lstm_cnn.hdf5
Epoch 5/50
7352/7352 [==============================] - 80s 11ms/step - loss: 0.3505 - acc: 0.8575 - val_l

Epoch 00005: val_acc improved from 0.79844 to 0.82117, saving model to weights_lstm_cnn.hdf5
Epoch 6/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.2938 - acc: 0.8829 - val_l

Epoch 00006: val_acc improved from 0.82117 to 0.85171, saving model to weights_lstm_cnn.hdf5
Epoch 7/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.2477 - acc: 0.9013 - val_l

Epoch 00007: val_acc improved from 0.85171 to 0.89141, saving model to weights_lstm_cnn.hdf5
Epoch 8/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.2269 - acc: 0.9119 - val_l

Epoch 00008: val_acc did not improve from 0.89141
Epoch 9/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1768 - acc: 0.9336 - val_l

Epoch 00009: val_acc improved from 0.89141 to 0.89549, saving model to weights_lstm_cnn.hdf5
Epoch 10/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1713 - acc: 0.9372 - val_l

Epoch 00010: val_acc improved from 0.89549 to 0.89786, saving model to weights_lstm_cnn.hdf5
Epoch 11/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1517 - acc: 0.9418 - val_l

Epoch 00011: val_acc improved from 0.89786 to 0.90499, saving model to weights_lstm_cnn.hdf5
Epoch 12/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1455 - acc: 0.9461 - val_l

Epoch 00012: val_acc improved from 0.90499 to 0.91517, saving model to weights_lstm_cnn.hdf5
Epoch 13/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1337 - acc: 0.9461 - val_l

Epoch 00013: val_acc did not improve from 0.91517
Epoch 14/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1417 - acc: 0.9418 - val_l

Epoch 00014: val_acc did not improve from 0.91517
Epoch 15/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1473 - acc: 0.9436 - val_l
```

```
Epoch 00015: val_acc did not improve from 0.91517
Epoch 16/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1467 - acc: 0.9414 - val_

Epoch 00016: val_acc did not improve from 0.91517
Epoch 17/50
7352/7352 [==============================] - 80s 11ms/step - loss: 0.1353 - acc: 0.9478 - val_

Epoch 00017: val_acc did not improve from 0.91517
Epoch 18/50
7352/7352 [==============================] - 81s 11ms/step - loss: 0.1250 - acc: 0.9479 - val_

Epoch 00018: val_acc improved from 0.91517 to 0.91720, saving model to weights_lstm_cnn.hdf5
Epoch 19/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1260 - acc: 0.9483 - val_

Epoch 00019: val_acc did not improve from 0.91720
Epoch 20/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1152 - acc: 0.9523 - val_

Epoch 00020: val_acc did not improve from 0.91720
Epoch 21/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1127 - acc: 0.9562 - val_

Epoch 00021: val_acc improved from 0.91720 to 0.92263, saving model to weights_lstm_cnn.hdf5
Epoch 22/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1216 - acc: 0.9486 - val_

Epoch 00022: val_acc did not improve from 0.92263
Epoch 23/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1122 - acc: 0.9516 - val_

Epoch 00023: val_acc did not improve from 0.92263
Epoch 24/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1267 - acc: 0.9460 - val_

Epoch 00024: val_acc improved from 0.92263 to 0.92297, saving model to weights_lstm_cnn.hdf5
Epoch 25/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1237 - acc: 0.9453 - val_

Epoch 00025: val_acc did not improve from 0.92297
Epoch 26/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1083 - acc: 0.9528 - val_

Epoch 00026: val_acc did not improve from 0.92297
Epoch 27/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1054 - acc: 0.9546 - val_
```

```
Epoch 00027: val_acc improved from 0.92297 to 0.93688, saving model to weights_lstm_cnn.hdf5
Epoch 28/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1503 - acc: 0.9476 - val_

Epoch 00028: val_acc did not improve from 0.93688
Epoch 29/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.2888 - acc: 0.9346 - val_

Epoch 00029: val_acc did not improve from 0.93688
Epoch 30/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.1738 - acc: 0.9440 - val_

Epoch 00030: val_acc did not improve from 0.93688
Epoch 31/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.1086 - acc: 0.9543 - val_

Epoch 00031: val_acc did not improve from 0.93688
Epoch 32/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0991 - acc: 0.9559 - val_

Epoch 00032: val_acc did not improve from 0.93688
Epoch 33/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.0972 - acc: 0.9570 - val_

Epoch 00033: val_acc did not improve from 0.93688
Epoch 34/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0906 - acc: 0.9591 - val_

Epoch 00034: val_acc did not improve from 0.93688
Epoch 35/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0937 - acc: 0.9587 - val_

Epoch 00035: val_acc did not improve from 0.93688
Epoch 36/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.0867 - acc: 0.9576 - val_

Epoch 00036: val_acc did not improve from 0.93688
Epoch 37/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0861 - acc: 0.9592 - val_

Epoch 00037: val_acc did not improve from 0.93688
Epoch 38/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0805 - acc: 0.9612 - val_

Epoch 00038: val_acc did not improve from 0.93688
Epoch 39/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0812 - acc: 0.9599 - val_
```

```
Epoch 00039: val_acc did not improve from 0.93688
Epoch 40/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.0814 - acc: 0.9610 - val_

Epoch 00040: val_acc did not improve from 0.93688
Epoch 41/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0778 - acc: 0.9650 - val_

Epoch 00041: val_acc did not improve from 0.93688
Epoch 42/50
7352/7352 [==============================] - 79s 11ms/step - loss: 0.0816 - acc: 0.9596 - val_

Epoch 00042: val_acc did not improve from 0.93688
Epoch 43/50
7352/7352 [==============================] - 81s 11ms/step - loss: 0.0757 - acc: 0.9646 - val_

Epoch 00043: val_acc did not improve from 0.93688
Epoch 44/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.0717 - acc: 0.9645 - val_

Epoch 00044: val_acc did not improve from 0.93688
Epoch 45/50
7352/7352 [==============================] - 81s 11ms/step - loss: 0.1078 - acc: 0.9610 - val_

Epoch 00045: val_acc did not improve from 0.93688
Epoch 46/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.1347 - acc: 0.9532 - val_

Epoch 00046: val_acc did not improve from 0.93688
Epoch 47/50
7352/7352 [==============================] - 78s 11ms/step - loss: 0.1108 - acc: 0.9638 - val_

Epoch 00047: val_acc did not improve from 0.93688
Epoch 48/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1112 - acc: 0.9637 - val_

Epoch 00048: val_acc did not improve from 0.93688
Epoch 49/50
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1112 - acc: 0.9648 - val_

Epoch 00049: val_acc did not improve from 0.93688
Epoch 50/50
7352/7352 [==============================] - 92s 12ms/step - loss: 0.1022 - acc: 0.9625 - val_

Epoch 00050: val_acc did not improve from 0.93688
```

```
In [71]: model.load_weights('weights_lstm_cnn.hdf5')
```

```
In [72]: # Confusion Matrix
         print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  0      405        61        0                   0
STANDING                 0       58       474        0                   0
WALKING                  0        0         0      479                  14
WALKING_DOWNSTAIRS       0        0         0        0                 416
WALKING_UPSTAIRS         0        0         0        7                  14

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                           25
STANDING                           0
WALKING                            3
WALKING_DOWNSTAIRS                 4
WALKING_UPSTAIRS                 450
```

```
In [77]: %matplotlib notebook
         import matplotlib.pyplot as plt
         import numpy as np
         import time
         # https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
         # https://stackoverflow.com/a/14434334
         # this function is used to update the plots for each epoch and error
         def plt_dynamic(x, vy, ty, ax, colors=['b']):
             ax.plot(x, vy, 'b', label="Validation Loss")
             ax.plot(x, ty, 'r', label="Train Loss")
             plt.legend()
             plt.grid()
             fig.canvas.draw()
```

```
In [78]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         fig,ax = plt.subplots(1,1)
         ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

         # list of epoch numbers
         x = list(range(1,50+1))

         # print(history.history.keys())
```

```
    # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
    # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, 

    # we will get val_loss and val_acc only when you pass the paramter validation_data
    # val_loss : validation loss
    # val_acc : validation accuracy

    # loss : training loss
    # acc : train accuracy
    # for each key in histrory.histrory we will have a list of length equal to number of 

    vy = history1.history['val_loss']
    ty = history1.history['loss']
    plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.3430146389593015
Test accuracy: 0.9368849677638276


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

15.2 Procedures:

- The necessary features were provided for classic ML algo which gave a 96% accuracy
- In LSTM+convnet without the feature engineering an accuracy ~94% was obtained